Self-Managed Cell: A Middleware for Managing Body-Sensor Networks

Sye Loong Keoh[1], Naranker Dulay[1], Emil Lupu[1], Kevin Twidle[1], Alberto E. Schaeffer-Filho[1], Morris Sloman[1], Steven Heeps[2], Stephen Strowes[2] and Joe Sventek[2]

[1]Department of Computing, Imperial College London, South Kensington Campus, London SW7 2AZ, UK
{slk, n.dulay, e.lupu, k.twidle, aschaeff, m.sloman }@imperial.ac.uk
[2]Department of Computing Science, University of Glasgow, 17, Lilybank Gardens, Glasgow G12 8RZ, UK
{heeps, sds, joe}@dcs.gla.ac.uk

Abstract— **Body sensor networks consisting of low-power on-body wireless sensors attached to mobile users will be used in the future to monitor the health and well being of patients in hospitals or at home. Such systems need to adapt autonomously to changes in context, user activity, device failure, and the availability or loss of services. To this end, we propose a policy-based architecture that uses the concept of a Self-Managed Cell (SMC) to integrate services, managed resources and a policy interpreter by means of an event bus. Policies permit the declarative specification of adaptation strategy for self-configuration and self-management. We present the design and implementation of the SMC and describe its potential use in a scenario for management of heart monitoring. Preliminary performance measurements are also presented and discussed.**

Keywords - autonomic management; adaptive sensing; policy-based adaptation; reconfigurable networks.

I. INTRODUCTION

There is an emerging trend in the healthcare industry to apply the advances in ubiquitous computing and sensor technology to provide continuous monitoring of a patient's medical condition anytime and anywhere. Numerous sensors and actuators have been developed for monitoring physiological parameters including pulse, heart-rate, oxygen saturation, as well as behavioural parameters such as posture and gait [1]. These sensors and actuators, which are usually wearable and often implantable can form a body sensor network using wireless communications. They can also interact with wearable processing units such as PDAs, mobile phones, and the fixed network infrastructure in the environment, thus providing a platform for continuous monitoring of the patient's vital signs. The body sensor network is particularly useful for monitoring patients in post-operative care or with episodic manifestations (such as cardiac arrhythmia). The benefits to the patients include early release from hospital, and early detection of abnormal conditions. Automated alerts to healthcare personnel can also be used to obtain help in the event of significant deviations from the norm or medical emergencies [2].

Wearable sensors without unwieldy wires between sensors and processing units allow patients to escape from being confined to a room, and enable them to be monitored while performing normal daily activities. Many current portable body sensors only collect information for offline analysis and are not integrated into a system to respond or adapt to patient activity. It is essential to provide real-time analysis of measurements to support adaptation according to the changes in context and clinical condition over time. Physiological parameters for clinical observation and monitoring, such as heart rate thresholds, and sensor configuration may need to be dynamically modified depending on the user's context, e.g., location, current activity and medical history. Devices should also adapt the frequency of measurements depending on the activity and clinical condition of the patient, thus optimising power consumption whilst ensuring that important episodes are not missed. Therefore, developing the architecture, tools and techniques which permit these environments to become self-managing is essential.

In this paper, we present a policy-based architecture that supports autonomic management and self-configuration for such systems, using the concept of a Self-Managed Cell (SMC) [3, 4]. An SMC is an architectural pattern that consists of an autonomous set of hardware and software components that represent an administrative domain. SMCs are able to function autonomously and adapt automatically to the user's current activity, communication capability and interact with other SMCs. A policy-based approach defines how the system should adapt in response to events such as failures, changes of context or changes in the requirements. This enables the SMC to implement a local feedback control loop over the system where changes of state in the managed objects and resources trigger adaptation that in turn affects the state of the system.

The paper is organised as follows: Section II describes the SMC architectural pattern and its main components. Section III describes the wireless medical and context sensors for health monitoring. We give an account of implementation details in Section IV while Section V discusses the initial measurements and performance evaluation of the prototype. Section VI and VII present the related work and conclude the paper with directions for future work.

II. THE SELF-MANAGED CELL (SMC)

Figure 1 illustrates the SMC architectural pattern where an SMC [3, 4] manages a set of heterogeneous components (i.e., managed resources) such as those in a body-sensor network, a room or even a large-scale distributed application. Resource adapters are instantiated to provide a unified view for interaction with managed resources as they may use different interfaces for communication and invocation of management actions.

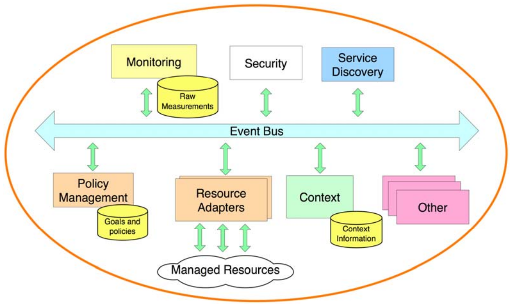


Figure 1. The SMC architecture pattern

The *event bus*, the *policy service,* and the *discovery service* detailed below constitute the SMC's core functionality and must be present in every SMC instantiation. However, SMCs may also comprise additional services for detecting context changes, monitoring of component behaviour and providing security (e.g., authentication and intrusion detection).

A. The Discovery Service

The discovery service is responsible for detecting new sensors, devices or other SMCs in the vicinity. It is responsible for maintaining the membership of the SMC to cater for transient failures, and permanent departure (i.e., switched off, out of range and failures). The discovery service also carries out any admission control for accepting the device into the SMC based on the device's profile and authentication information available. By making the discovery service policy-driven, it can easily be adapted to different applications.

The discovery service broadcasts its identity message (id; type[; extra]) at frequency ω_R. This enables the SMC to advertise itself to both devices and other SMCs and enables current SMC members to determine that they are still within reach of the SMC. New devices respond to the identity message with a unicast message identifying themselves. The discovery service can then query the device to obtain a device profile and authentication credentials, and decides whether to accept the device for membership and to which role it should be assigned. An accepted device is informed and a *component-detected* event is generated which results in the device being registered and assigned to a role in the SMC. A device specific communication adapter is then also created for that device. Each existing member device unicasts its identity message to the discovery service at the frequency ω_D. If the discovery service misses n_D successive messages from a particular device, it concludes that the device has left the SMC permanently, and generates a corresponding *component-left* event. This event will trigger the removal of any adapters corresponding to that device in the policy service.

B. The Event Bus

Ubiquitous systems are essentially event driven, as changes of state in resources need to be notified asynchronously to several, potentially unknown services. The event bus distributes events to software and hardware components within the SMC. It offers at-most-once, persistent delivery of events and implements content-based filtering [5]. SMC events comprise (attribute, value) pairs and our implementation supports the specification of filters (i.e., constraints on the attributes and their values) e.g., heart-rate > 110. Subscribers register to receive notifications based on filters and events matching a registered filter are forwarded to the appropriate subscribers.

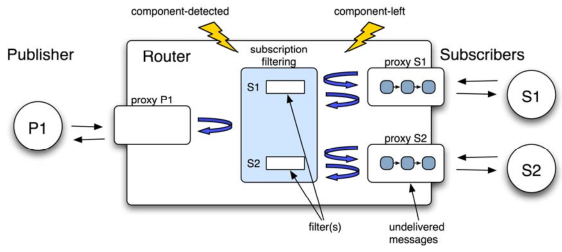


Figure 2. The event bus architecture

The event bus must guarantee reliable event delivery since events trigger adaptation and re-configuration actions. The publisher ensures that events it publishes arrive at the event bus, while each subscriber is guaranteed to receive events in the same order as received by the event bus. This is required as events from the same publisher may be causally related. Thus, all messages are acknowledged when received by the event bus and the subscribers.

The architecture of the event bus is shown in Figure 2. On receiving a *component-detected* event from the discovery service, the event bus creates a proxy for that new component. Event occurrences are notified by publishers to the event bus via their proxies. Each subscriber proxy maintains a FIFO queue of events and attempts to deliver the event at the head of the queue until it is successful. When a component is detected to have left the SMC, a *component-left* event will be raised by the discovery service, which causes the event bus to remove that subscriber's filters, and purge any queued up events for that subscriber.

C. The Policy Service

Policies are rules that govern choices in the behaviour of an SMC. The use of interpreted policies means that SMC behaviour can be easily changed without shutting down or re-coding components. We are primarily concerned with two types of policies: authorisation policies define what actions are permitted under given circumstances and obligation policies define what actions to carry out when specific events occur if certain conditions are fulfilled (Event-Condition-Action rules).

The policy service [4, 6] maintains adapter objects for each of the components on which management actions can be performed. This includes the sensors and other devices present in the SMC, services within those devices and remote SMCs. These adapter objects (also called managed objects) are grouped in a domain structure that implements a hierarchical namespace. Domains may overlap and a managed object may belong to several domains. Domains are typically used to group objects to which common policies apply and can be used as placeholders in policy specifications. Domains and policies are managed objects in their own right on which actions can be performed e.g., adding/removing an object from a domain, enabling or disabling a policy. Discovery obligation policies triggered by a *component-detected* event create the adapter object to interact with the discovered object and determine in which domain the object will be placed. Other policies specified for that domain will then automatically apply to the new component.

III. Wireless Medical and Context Sensors

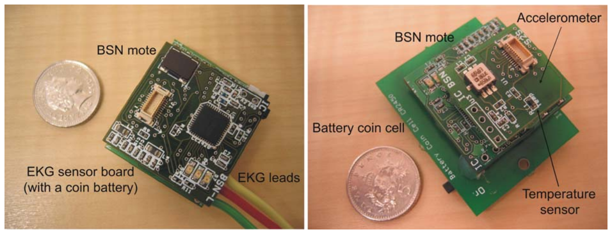
In this section, we detail various mote-based medical and context sensors used in our prototype implementation.

A. *Mote-based Electrocardiograph (EKG)*

Electrocardiograph (EKG) is used to measure the electrical activity of the heart in which it records a short sampling of the heart's electrical activity between different pairs of electrodes. Each pair of leads provides unique and detailed information of the cardiac rhythm. This enables the cardiologist to rapidly identify a wide range of cardiac arrhythmias, acute myocardial ischemia and infarction. Mote-based EKGs [7] are wearable and they provide continuous EKG monitoring during daily activities. Figure 3(a) shows an EKG sensor attached to the BSN that is used in our prototype implementation. It is a three-lead EKG sensor in which two leads attached to the upper chest measures the cardiac activity, while the other serves to properly bias the patient's skin. The resulting trace is routed to an analogue to digital converter (ADC) port on the BSN mote [8].

B. *Accelerometer and Temperature Sensor*

The accelerometer provides information on the activities of the wearer, e.g., walking, sleeping, running or resting. Detection of a total lack of movement, combined with abnormal sensor readings, can be used to infer if the patient has fallen or the sensors have been removed.



(a) EKG (b) Accelerometer and temperature

Figure 3. Wireless medical and context sensors

Figure 3(b) shows a 2-axis accelerometer that can be attached to the BSN mote. Similar to the work reported in [9], the accelerometer is used for simple activity analysis in which we are only interested in the level of activity without being able to distinguish specific actions. Strenuous activities such as walking and running can be detected based on a characteristic periodic acceleration frequency signature indicating the speed of movement [9].

The temperature sensor is placed on the same sensor board as the accelerometer. However, it can only be used to sense environmental, not body temperature. The sensor must be calibrated before use.

IV. Prototype Implementation

A. *Prototype Scenario*

We have developed a prototype to monitor the heart-rate of an elderly patient who suffers from heart disease, so that any potential heart-attack can be detected. We model the scenario as an SMC consisting of a Gumstix running the SMC core services, together with an EKG sensor, accelerometer, temperature sensor, a GPRS-enabled mobile phone and a GPS.

The SMC is instantiated by starting the policy service, which in turns instantiates the event bus and the discovery service as managed objects. Policies facilitating self-management and self-configuration to support heart monitoring are pre-deployed in the policy service. This includes discovery policies to create managed objects to communicate with sensors and place them in the appropriate domain. We assume that *factories,* which permit the creation of new managed objects for the various SMC sensors have already been imported in the `/factories/` domain. The discovery service discovers various sensors over 802.15.4, Wi-fi and Bluetooth.

Policies are defined to alert the hospital if a heart problem is detected. The following policy specifies that when the EKG sensor raises an event that the patient's heart rate has exceeded 90 bpm while resting, it triggers an action to send an emergency SMS to the hospital to call for an ambulance. If the patient is located outdoors, it is also possible to include the patient's current GPS location when indicating an emergency.

```
on heart_attack_event(hr) do
    loc = /devices/gps.getLocation();
    /devices/mobilephone.call(2075948449, 2025487584, hr, loc);
```

EKG reconfiguration can be performed in order to adapt the heart rate threshold whenever there is a change in the patient's current activity. The maximal heart-rate is calculated by subtracting the patient's age in years from the general maximal heart rate of 220 bpm. During exercise, the heart-rate should reach no more than 60 – 75% of the maximal heart-rate calculated. The following policies are defined:

```
on strenuous_event(age) do
    /devices/ekg.setThreshold(0.6*(220-age));
```

B. *Implementation of the SMC Core Services*

We have implemented the basic architecture of the SMC consisting of a policy service, event bus and discovery service using Java 1.4 and Java 2 Micro Edition (J2ME). We have deployed the SMC core services on a Gumstix running Linux.

The policy service has been implemented with a particular focus on flexibility and the ability to load all the code needed on-demand. This enables us to use it across a wide variety of applications and devices with different capabilities by only loading components necessary for that particular application. The policy service contains several factory objects such as policy (authorisation and obligation), domain, managed object, and event. This provides the flexibility to dynamically create policies, managed objects or adapter objects for communicating with various sensors and devices with their respective communication protocol such as Bluetooth, 802.15.4, or Zigbee. Specific factories can be defined for each of the different types of managed objects in use, e.g., an EKG factory creates a BSN managed object that can communicate via 802.15.4 with the EKG mote. The created instance of the EKG mote is then placed in the domain structure.

C. Implementation and Configuration of Sensors

We have implemented a generic architecture for BSN motes (running TinyOS and programmed using nesC [10]) that enables the sensors to be discovered dynamically and to publish events to the event bus in the SMC. Different sensors are customised to sense different physiological parameters based on the sensors attached to the BSN mote.

We have implemented an algorithm based on [11] to detect QRS complex and measure RR interval from the EKG waveform on the BSN mote. The algorithm first performs differentiation over the sampled data in order to obtain information about signal *slope*. A potential peak is characterised by a point where the value of the slope is zero [12]. The value of the derivative is then squared to intensify the frequency response curve of the derivative. Finally, the algorithm performs a running integration of the squared derivative over a moving window. This value is then compared with an adaptive threshold, if it is exceeded, an R wave onset is assumed. The R peak value is then stored in order to calculate RR intervals. Based on the mean RR intervals, heart-rate can be derived and then be sent as an SMC event to the event service if it exceeds the threshold. The EKG data is sampled at 200 Hz and the QRS detection algorithm is executed every time a sample is obtained. Our implementation has the advantage of optimising the sensor's battery power as it does not transmit all EKG data to another device for processing and analysis.

The accelerometer and temperature sensor data are obtained from the respective ADC ports for analysis. Whenever there is a change in the activity, i.e., from resting to walking or running and vice versa, an SMC event is published by the sensor.

V. Evaluation and Measurements

This section presents an initial evaluation of the SMC's performance based on a Gumstix device running the core services to manage a set of BSN medical sensors.

A. The Discovery Service

Although 802.15.4 radio specification claims a maximum bandwidth of 250 Kbps, the actual bandwidth measured in our experiments is substantially lower. We conducted an experiment comprising a single BSN node as a receiver and several BSN nodes acting as senders. Each sender was one hop away from the receiver and each sent a packet of 76 bytes with a data rate progressively increasing from 1 to 40 packets per second. Figure 4 shows that the data throughput is linear with the data rate for one sender. In the case of 3 and 5 senders, a maximum throughput of approximately 50 Kbps is observed.

We also measured the end-to-end delay of the 802.15.4 radio between two BSNs that are one hop away from each other. It takes approximately 20 ms to send a packet to another BSN. The end-to-end delay of the serial link between the Gumstix and the BSN gateway is approximately 25 ms. As described earlier, the discovery handshake involves three 802.15.4 packets and two serial packets which amount to 110 ms, the actual delay measured was on average 129.40 ms.

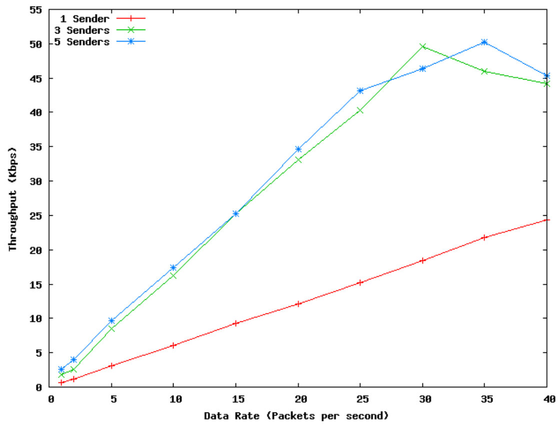


Figure 4. Throughput of the Body Sensor Nodes (BSN) with varying data rates

B. The Policy Service

We observed that the evaluation of policy constraints incurs the most overheads as this involves parsing of the constraints and string comparisons. The time taken to execute a policy without a condition and with an empty action is only 13.57 ms, while it takes 28 ms (variance is 0.49) to execute a policy with a simple condition and an action to publish a new event. We also observed that it takes 23.88 ms with a variance of 0.38 to execute a policy (with no condition) and invoking an action to issue a command to the BSN.

Each policy is instantiated as a Java object which requires 3.214 KB memory. This includes the policy type, the list of events which may trigger the policy, the actions to be performed and the constraints that need to be evaluated.

C. The Event Bus

We measured the performance of the event bus by measuring the delay incurred from the moment an event is raised until the notification is received by the subscribers as a function of number of subscriptions. The event bus is very efficient in that the average time to match an event against various number of subscriptions (1 to 1000 subscriptions) is between 13 to 15 ms. This performance is adequate for the case of an SMC that manages a few body sensors, forming a body area network.

D. End-to-end Management

The whole discovery process takes an average of 144.25 ms. This is measured from the time a device sends a request to join the SMC, performs the discovery protocol handshake, generates a new member event to instantiate the managed object and event proxy. The process of creating the managed object and the event proxy takes approximately 14.85 ms.

We also measured the time from invocation of an action until it returns. This involves issuing a command to the BSN via the 802.15.4 transport, e.g., to instruct the BSN to start sensing. The result shows that it takes an average of 12.7 ms. As for end-to-end management, when the sensor publishes an event through the event proxy, which then triggers the policy to execute a management action, it takes an average of 46.05 ms. This was measured from the time the event arrived at the event proxy at the event bus till the action in the policy was executed.

VI. Related Work

Traditional approaches in policy based network and systems management include PCIM [13], PDL [14], NGOSS Policy [15], Ponder [16] and PMAC [17]. They all make use of event-condition-action rules for adaptation but are aimed at the management of distributed systems and network elements and do not scale down to small devices and sensors.

Gaia [18] and Aura [19] introduce active space and smart space respectively to provide a "meta-operating system" to build pervasive applications. They focus on spaces of relatively fixed size, and on specific concerns such as context-related applications and user presence. We consider a SMC as an architectural pattern that applies at different levels of scale and we focus on generic adaptation mechanisms through policies.

PICO [20] is a middleware that enables effective collaboration among heterogeneous hardware and software entities that work together to achieve goals. The notion of community is similar to our SMC, but our focus is to facilitate self-configuration and self-management using policies.

CodeBlue [21] integrates low-power, wireless vital sign sensors, PDAs and PC-class systems to provide a platform for medical sensor networks. It investigates the data rates, patterns of packet loss and route maintenance of the wireless sensor network, while the SMC focuses on the management of body-sensor networks using policies.

Mitidieri and Kaiser [22] introduced a filtering mechanism for a publisher/subscriber communication system. However, it seems that by requiring an event handler on each BSN sensor to manage event subscriptions is not feasible in our framework.

VII. Conclusions and Future Work

We have proposed the SMC structure as a basic architectural pattern that aims to provide local feedback control and autonomy. Our implementation demonstrates that the SMC concept can be applied to e-Health applications in order to achieve self-management and self-configuration in body sensor networks. The realisation of more complex systems through the composition and peer-to-peer interactions between SMCs is required to cater for a larger scale of ubiquitous applications.

The use of an event bus as the primary means of exchanging management information de-couples architectural components and provides the basis for extending the functionality of the SMC by adding additional services. Policies, in the form of ECA rules, provide a simple and effective encoding of the adaptation strategy required in response to changes of context or changes in requirements. The ability to dynamically load, enable and disable the policies together with the ability to use policies in order to manage policies caters for a wide variety of application needs.

Preliminary performance measurements suggest that the delays incurred are suitable for many health-monitoring applications for management of chronic conditions and in post-operative care. This is despite the fact that the current implementation focuses on flexibility rather than performance and no optimisations have been made.

Acknowledgment

The authors wish to thank the UK Engineering and Physical Sciences Research Council (EPSRC) for their support of this research through grants GR/S68040/01 and GR/S68033/01.

References

[1] G.Z.Yang (Ed.), Body Sensor Networks, Springer-Verlag, March 2006.

[2] E.Jovanov, *et al.* A Wireless Body Area Network of Intelligent Motion Sensors for Computer Assisted Physical Rehabilitation. *Journal of NeuroEngineering and Rehabilitation* 2(6), March 2005.

[3] N. Dulay *et al.* Self-Managed Cells for Ubiquitous Systems, *In the Proc. of the 3rd Int. Conf. on Mathematical Methods, Models and Architectures for Computer Networks Security*, Sept 2005.

[4] S.L. Keoh, *et. al.* Policy-based Management for Body-Sensor Networks. In *Proc. of the 4th Int. Work. on Wearable and Implantable Body Sensor Networks (BSN),* 26 – 28 Mar 2006, Aachen, Germany.

[5] S. Strowes, *et al.* An Event Service Supporting Autonomic Management of Ubiquitous Systems for e-Health, In *Proc. of the 5th Int. Work. on Distributed Event-based Systems*, Lisbon, Portugal, July 2006.

[6] Ponder2, http://www.ponder2.net/

[7] B.Lo, *et al.* Body Sensor Network – A Wireless Sensor Platform for Pervasive Healthcare Monitoring. In *Adjunct Proc. of Int. Conf. on Pervasive Computing,* 2005.

[8] U.Anliker, *et al.*. AMON: A Wearable Multiparameter Medical Monitoring and Alert System. In *IEEE Trans. on Information Technology in Biomedicine 8(4)*, Dec 2004.

[9] D.Gay, *et al.* The nesC Language: A Holistic Approach to Networked Embedded Systems. In *Proc. of Programming Language Design and Implementation (PLDI),* San Diego, June 2003.

[10] DTI UbiMon Project, UK www.ubimon.org

[11] J.Pan and W.J.Tompkins. A Real-time QRS Detection Algorithm. *IEEE Transactions in Biomedical Engineering,* 32, 1985.

[12] E. Katsiri, *et al.* Embedded Real-Time Heart Variability Analysis. In *Proc. of the 4th Int. Work. on Wearable and Implantable Body Sensor Networks (BSN),* 26 – 28 Mar 2006, Aachen, Germany.

[13] B. Moore, *et al.* Policy Core Information Model Version 1 Specification, Network Working Group, RFC2060, 2001

[14] J. Lobo, *et al* A Policy Description Language. In *Proc. of the 16th National Conference on Artificial Intelligence,* Orlando, July 1999.

[15] J. Strassner, *Policy-based Network Management*, 2004.

[16] N. Damianou, *et al*. The Ponder Policy Specification Language. In *Proc. of the Int. Workshop on Policies for Distributed Systems and Networks (POLICY),* Bristol, UK, Jan 2001.

[17] D. Agrawal, *et al.* Policy Management for Networked Systems and Applications, In *Proc. of the 9th IFIP/IEEE Int. Symp. on Integrated Network Management,* Nice, France, May 2005.

[18] M. Roman, *et. al.* A Middleware Infrastructure for Active Spaces, *IEEE Pervasive Computing,* 1(4):74-83, 2002.

[19] D. Garlan, *et al.* Aura: Toward Distraction-Free Pervasive Computing, *IEEE Pervasive Computing,* 1(2), 2002, pp. 22 - 31.

[20] M. Kumar, *et al.* PICO: A Middleware Framework for Pervasive Computing, *IEEE Pervasive Computing,* 2(3):72-79, 2003.

[21] D. Malan, *et al.* CodeBlue: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care. In *Proc. of the Int. Work. on Wearable and Implantable Body Sensor Networks*, April 2004.

[22] C. Mitidieri and J. Kaiser. Attribute-based filtering for Embedded Systems. In *Proceedings of the 2nd Int. Work. on Distributed Event-based Systems (DEBS),* June 2003